

REMARKS/ARGUMENTS

Applicants request the Examiner to reconsider this application in view of the claim amendments and the following remarks.

Pointer Table System

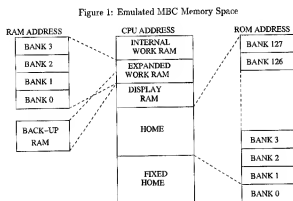
Applicants direct the Examiner's attention to the following claim limitations:

- Independent claim 6 as amended includes a limitation "wherein said video game software image comprises multiple ROM pages and said method further includes said emulator program providing a pointer table system that allocates ~~allocating~~ ROM pages in said target computing device read/write memory and ~~duplicating~~ duplicates at least a portion of said allocated ROM pages."
- Independent claim 14 includes a similar limitation in its last paragraph.
- Independent claim 37 recites in its last paragraph "third emulation program instructions that provide a pointer table system which allocates ROM pages in target platform read/write memory and duplicates at least a portion of said allocated ROM pages to facilitate paging operations."
- Dependent claim 23 further recites "using a page-said pointer table system to control memory access by remapping memory access instructions into different memory locations and/or function calls."
- Dependent claim 24 recites, in combination, "providing a read only memory protection function to eliminate overwriting of read only memory."

Applicants believe these features, in combination, each patentably define the claimed subject matter over the prior art.

Applicants do not broadly recite a paging architecture, duplication of ROM pages. caching, or providing write protection. Rather, applicants' claim recites a unique implementation of a pointer table system.

Figure 1 set forth below shows an example memory map of the CGB system, of which the "CPU ADDRESS" space is 64k (Not including any additional RAM/ROM memory that can be accessed by bank switching). Fig. 15 in the original patent application shows a double pointer system for both the read and write pointers for the "CPU ADDRESS" space. This is a unique setup as each single addressable byte of memory is represented by plural (e.g., four) pointers:



As shown in figure 2, using this exemplary pointer system the applicants are able to implement a unique method of ROM protection on read only memory. By the write table to point to NULL or dummy memory, applicants can effectively prevent writes to areas of "read only memory" that is being emulated by random access memory. In figure 3, using this plural pointer system, the read and write table both point to the same data allowing a fast lookup for the data for the read/write operations:

Figure 2: MBC Read-Only Memory

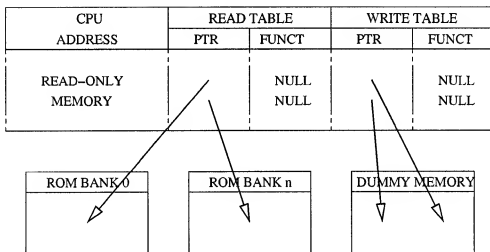


Figure 4: MBC HW I/O Memory

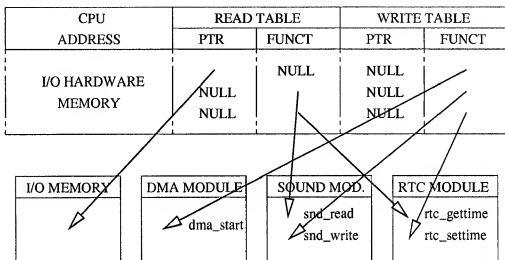


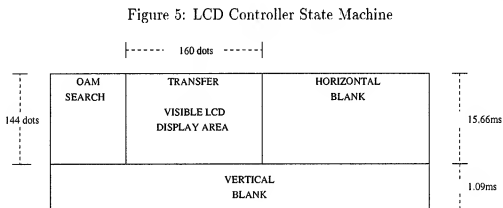
Figure 4 shows several different examples of I/O memory. The top could for example represent an emulated DMA device, where the write table points to the dma_start function, which is called upon a write. The read table is pointing to memory, which indicates any relevant reads can be read directly from the memory without the overhead of running through a function call. The bottom two could represent emulated sound and real time clock devices. For these examples both the read table and write table are pointing to the corresponding read and write function handlers for these hardware devices.

This feature as recited in the above-mentioned claims is not taught or suggested by the applied references. The Examiner observes that Snex9X is silent regarding ROM pages, but relies on Dahl to supply the missing teachings. However, Dahl does not teach or suggest the paging table system as now claimed.

LCDC Timing

Each of the independent claims herein further requires, in combination, a target computing device to model display timing activities of a handheld computing device. See also dependent claim 25 ("The method of claim 6 wherein said modeling includes using a state machine defining a horizontal blank state and a vertical blank state") and dependent claim 26 ("providing a cycle timer to determine when a modeled state has expired and transition to a new state is desired.") These features in combination further define the respective claims over the prior art.

Fig. 8 and Fig. 9A from the original patent application show creative use of a "LCD Controller State Machine" to drive the timing of the emulated CPU processor. Figure 5 shows a graphics representation of this:



In this picture you can clearly see the OAM SEARCH (also referenced as DEFINING SEARCH in the original patent application), OAM TRANSFER, HBLANK, and VBLANK states in relation to the visible dots (pixels) on the LCD display.

A typical emulator will drive its timing off of the number of clock ticks to emulate a specific instruction and then emulate any required hardware. In contrast, applicants'

illustrative non-limiting disclosed implementation drives events based on the state progression of the "LCD Controller State Machine" [0086]. This state machine drives events such as emulated interrupts, display updates, emulated devices such as the sound device, dma devices, and external inputs such as user handset events.

Such a feature as recited in the above-mentioned claims is not taught or suggested by the applied references. The Examiner asserts that the SNES9x emulator discloses this feature at page 8. However, applicants do not find on that page any disclosure of the handheld controller state feature now claimed.

Flashing Object Pattern Correction

Amended dependent claim 27 recites, in combination, "selectively skipping frames while maintaining execution of instructions to maintain state information while minimizing game play slowdowns." Games designed for a specific target platform will often take advantage of the decay rate of the LCD screen. For example, a game will change the on/off state of an object at every frame to create a semi-transparent effect. See figure 6 and figure 7.

Figure 6: Object blinking at 15fps













Game Frame No	0	1	2	3	4	5	6	7	8	9	10	11
Visible Object												

Figure 7: Object blinking at 30fps

Game Frame No	0	1	2	3	4	5	6	7	8	9	10	11
Visible Object												

On a target platform that has sufficient CPU power, this is not a major issue.

Figure 8 shows a system that has a screen display running at 85Hz and is sufficiently fast so that the emulation does not have to skip rendering any frames. As shown in the figure, the frames shown on the screen do not directly mirror the frames rendered by the emulator due to the difference in the refresh rate of the two systems. On the emulated system it is blinking the object every other frame; while on the emulator screen you end up with a more interesting pattern displayed. You will also observe the displayed frames slightly lag what is on the emulated screen.

Figure 8: Object blinking at 30fps on a 85Hz display with no dropped frames

Game Frames		1		3		5		7		9		11			
Display Frames	0	1		3	4		6	7		9		12		15	

A modern computer display such as a LCD or CRT has a higher response time then the LCD response time of the display from the emulated system. Thus, to properly emulate the emulated display - alpha blending between the current emulated frame and previous emulated frame is often applied. This allows the emulator to emulate the slow decay rate of the display from the emulated system, and provide a semi-transparent look to blinking objects while slightly blurring the display. This blurring also matches the blurring that occurs on the reference hardware due to the slow LCD decay rate on that hardware.

In preferred embodiment the emulator would maintain the sixty frames-per-second rate at all times, and alpha blend between the current and last frame for a best

case emulation of the reference hardware. However, it is not always possible to maintain a sixty frames-per-second rate on a slow target platform. /0191]

If a slow target platform were to render the graphics as shown in *figure 8* at 30fps you could end up with a "always on" or "always off" on pattern, as only the even (or odd) game frames would be rendered. Rather than an object appearing semi-transparent the object would appear solid or not at all. With "dynamic-scaling" you end up with strange moiré patterns with respect to the blinking objects. You can end up with an object that appears for a second, and then disappears for another second. This is a drastically different behavior from the intended semi-transparency. As different games have different patterns of blinking objects, we require a custom solution for each game to create the "best case" solution for each game while running on a slow platform. [01921]

The Examiner rejects claim 27 based on an assumption that "Snes9x discloses a frame skip count that enables the selectively skip frames (Snes9x page 8)". Snes9x discloses and uses a frame skip count that enables the user to selectively skip frames. Snes9x also discloses that this is "Ideal for some Super FX games that confuse the auto-adjust code or for games that deliberately flash the screen every alternate frame." This may be similar to Giles [0016] where the emulator can selectively skip rendering part of the graphics for increased performance at the loss of uncertain imaging results.

However, applicant's exemplary illustrative non-limiting implementation uses an index variable that is assigned by hand to each game's ROM image that indexes into a pattern lookup for which frames to render. This extends beyond the scope of Snes9x as applicants' non-limiting implementation does not need to be a regular or distinct pattern, allowing the emulator to display key frames in order to faithfully reproduce key effects

and blinking objects while dropping enough frames to real-time performance. This feature in combination is not taught or suggested by the applied references.

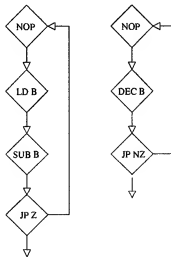
"No Operation" Loop Reduction

Amended claim 28 recites, in combination, "further including performing no-operation look-ahead to avoid wasting processing time in no-operation loops."

The original patent application in [0196] states:

Some games and other applications make extensive use of "no operation" loops to maintain game timing. Somewhat surprisingly, such "no operation" loops can cause emulator to run very slowly. To avoid this particular issue, it is possible for emulator to include a dynamic code analyzer that "looks ahead" to the next few instructions surrounding a "no op" instruction to determine whether the game file includes a "no op" loop.

Figure 9: Some possible "no operation" loops



As shown in *figure 9* above, the emulator can use a dynamic code analyzer to "look ahead" at the next few instructions, and upon detecting a recognized pattern, can

skip emulating the entire loop. Time as emulated for the LCD display and any pertinent interrupts will need to be handled accordingly, but emulation of the loop can be completely bypassed.

Mullarkey 7:8-48 is using the look-ahead circuit to optimize speed and power usage by turning on and off memory banks. Applicants' technique is different. While both techniques use look-ahead for the following *n*. instructions, applicants' illustrative non-limiting technique differs by skipping the following instructions, and in turn the loop itself in its entirety. This diverges from the sense of strict emulation where each instruction is emulated and allows the CPU emulator to stop emulating until the emulated LCD or interrupt time has reached a point that would have caused the loop to terminate had it been emulated. This feature is not taught or suggested by the applied references.

All outstanding issues have been addressed and this application is in condition for allowance. Should any minor issues remain outstanding, the Examiner should contact the undersigned at the telephone number listed below so they can be resolved expeditiously without need of a further written action.

LINK
Appl. No. 10/690,818
April 24, 2007

Respectfully submitted,

NIXON & VANDERHYE P.C.

By: /Robert W. Faris/

Robert W. Faris
Reg. No. 31,352

RWF:ejs
901 North Glebe Road, 11th Floor
Arlington, VA 22203-1808
Telephone: (703) 816-4000
Facsimile: (703) 816-4100